

# A novel Hopfield neural network approach for minimizing total weighted tardiness of jobs scheduled on identical machines

Norbert FOGARASI

Department of Telecommunications;  
Budapest University of Technology and Economics  
Budapest, Hungary;  
email: [fogarasi@hit.bme.hu](mailto:fogarasi@hit.bme.hu)

Kálmán TORNAI

Faculty of Information Technology;  
Pázmány Péter Catholic University  
Budapest, Hungary;  
email: [tornai.kalman@itk.ppke.hu](mailto:tornai.kalman@itk.ppke.hu)

János LEVENDOVSKY

Department of Telecommunications;  
Budapest University of Technology and Economics  
Budapest, Hungary;  
email: [levendov@hit.bme.hu](mailto:levendov@hit.bme.hu)

**Abstract.** This paper explores fast, polynomial time heuristic approximate solutions to the NP-hard problem of scheduling jobs on  $N$  identical machines. The jobs are independent and are allowed to be stopped and restarted on another machine at a later time. They have well-defined deadlines, and relative priorities quantified by non-negative real weights. The objective is to find schedules which minimize the total weighted tardiness (TWT) of all jobs. We show how this problem can be mapped into quadratic form and present a polynomial time heuristic solution based on the Hopfield Neural Network (HNN) approach. It is demonstrated, through the results of extensive numerical simulations, that this solution outperforms other popular heuristic methods. The proposed heuristic is both theoretically and empirically shown to be scalable to large problem sizes (over 100 jobs to be scheduled), which makes it applicable to grid computing scheduling, arising in fields such as computational biology, chemistry and finance.

**Computing Classification System 1998:** G.1.6

**Mathematics Subject Classification 2010:** 90C27

**Key words and phrases:** scheduling theory, artificial neural networks, Hopfield neural network, total weighted tardiness problem, quadratic optimization

## 1 Introduction

With the advent of grid computing, the classical science of scheduling theory has gained a new sphere of applications. Methods which were originally developed for decision making around limited resources in manufacturing and service industries have been adopted in the areas of computer science, telecommunication and other computationally intensive disciplines such as computational biology, chemistry and finance [5, 22]. Specifically, in the area of computational finance, where this paper sources its motivation, the problems of portfolio selection, pricing and hedging of complex financial instruments requires an enormous amount of computational resources whose optimal usage is of utmost importance to investment banks. The prices and risk sensitivity measures of complex portfolios need to be reevaluated daily, for which an overnight batch of calculations is scheduled and performed for millions of financial transactions, utilizing thousands of computing nodes. Each job has a well-defined priority and required completion time for availability of the resulting figures to the trading desk, risk managers and regulators. The jobs can generally be stopped and resumed at a later point on a different machine which is referred to as *preemption* in scheduling theory. For simplicity of modeling the problem, machines are generally assumed to be *identical* and there is a known, constant number of machines available.

The problem of finding optimal schedules for jobs running on identical machines has been extensively studied over the last three decades. Sahni [25] presents an  $O(n \log mn)$  algorithm to construct a *feasible* schedule, one that meets all deadlines, if one exists, for  $n$  jobs and  $m$  machines. The basic idea of the algorithm is to schedule jobs with earliest due dates first, but fill up machines with smaller jobs if possible. Note that this method allows the development of an algorithm to compute the minimal amount of unit capacity for which a feasible schedule exists. This result has been extended to machines with identical functionality but different processing speed, termed *uniform machines*, and jobs with both starting times and deadlines [19]. However, the scheduling task becomes more difficult when a feasible schedule does not exist and the goal is to minimize some measure of delinquency, often termed *tardiness*. Tardiness of an individual job under a given schedule is defined as the amount of time by which the job finishes after its prescribed deadline, and is considered to be zero if the job finishes on or before the deadline.

In case of minimizing the maximum tardiness across all jobs, Lawler [14] shows that the problem is solvable in polynomial time, even with some precedence constraints. Martel [19] also used his construction to create a polynomial

time algorithm to find the schedule which minimizes maximum lateness. However, if our measure concerns the total tardiness instead of the maximal one, then even the single machine, total tardiness problem (without weights) was proven to be NP-hard by Du et al [7]. A pseudopolynomial algorithm has been developed by Lawler [13] for this problem, using dynamic programming, but this is for the 1-machine problem and does not have good practical runtime characteristics.

In practical applications, jobs often have relative priorities associated with them, represented by positive real *weights* and the objective becomes minimizing the total weighted tardiness (TWT). Once the NP-hardness of the TWT problem was established, most of the research work on the problem concerned the development of fast, heuristic algorithms. Dogramaci et al. [6] propose a simple heuristic for the total (non-weighted) tardiness problem without preemption. Rachamadugu et al. [23] then studied the identical machine, total weighted tardiness problem without preemption. They proposed a myopic heuristic and compared this to earliest due date (EDD), weighted shortest processing time (WSPT) and Montagnes rule on small problem sizes (2 or 5 jobs in total). Azizoglu et al. [3] worked on an algorithm to find optimal schedule for the unweighted total tardiness problem without preemption, but their branch and bound exponential algorithm is too slow, in practice, for problems with more than 15 jobs. Armentano et al. [2] examined the non-weighted problem without preemption, and starting from the KPM heuristic of Koulamas [11] improved upon it, using tabu search. Guinet [8] applies simulated annealing to solve the problem with uniform and identical machines and a lower bound is presented in order to evaluate the performance of the proposed method. More recently, Sen et al. [26] surveyed the existing heuristic algorithms for the single-machine total tardiness and total weighted tardiness problems while Biskup et al. [4] did this for the identical machines total tardiness problem and also proposed a new heuristic. Akyol et al. [1] provide an excellent recent review of artificial neural network based approaches to scheduling problems and proposes a coupled gradient network to solve the weighted earliness plus tardiness problem on multiple machines. The feasibility of the method is illustrated on a single 8-job scheduling problem.

We suggest a novel heuristic for the TWT problem, based on the Hopfield Neural Network approach which is shown to perform better than existing simple heuristics and has desirable scaling characteristics. Maheswaran et al. [16] applied a similar approach to the single machine TWT problem and their results were encouraging for a specific 10-job problem.

In this paper, we first map the problem into quadratic optimization and then

the Hopfield net is used to provide fast polynomial time, heuristic solution. The topics are organized as follows: in Section 2, we present the problem formulation and the model used, in Section 3 existing heuristics are defined and explained, in Section 4 our novel HNN approach is introduced, in Section 5 numerical results are presented and finally in Section 6 some conclusions are drawn and directions for future research are outlined.

## 2 Problem formulation

In this section, we give a formal presentation for the problem of optimally scheduling jobs on finite number of identical processors under constraints on the completion times. The basic formalism is the following:

- Given  $N$  jobs with sizes  $\mathbf{x} = \{x_1, x_2, x_3, \dots, x_N\} \in \mathbb{N}^N$ . The processing of the jobs can be stopped and resumed at any time, so the processing time units of each job need not be contiguous. In the literature this condition is known as preemption and also assumes a task started on one machine can continue on another [5].
- For each job a cutoff time is prescribed by  $\mathbf{K} = \{K_1, K_2, K_3, \dots, K_N\} \in \mathbb{N}^N$ . This constraint defines the time within which the job is to be completed.
- The constant number of processors, the capacity of the system is denoted by  $V \in \mathbb{N}$ .
- We are also given a vector  $\mathbf{w} = \{w_1, w_2, w_3, \dots, w_N\} \in \mathbb{R}^N, w_i \geq 0, \forall i = 1, \dots, N$  denoting the relative priority (or weight) of each job, which can be used in the definition of the objective function.

A schedule is represented by a binary matrix  $\mathbf{C} \in \{0, 1\}^{N \times L}$  where  $C_{i,j} = 1$  if job  $i$  is being processed at time slot  $j$ , and  $L$  denotes the length of the schedule. An example is given in (1) where the parameters are the following:  $V = 2, N = 3, \mathbf{x} = \{2, 3, 1\}, \mathbf{K} = \{3, 3, 3\}$ .

$$\mathbf{C} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad (1)$$

The first row in (1) denotes the fact that under this schedule  $\mathbf{C}$ , the first job is processed in time steps 1 and 3 (note that preemption is used as the processing of this job is not continuous) and therefore the prescribed size 2 of this job

completes within the prescribed cutoff time of time step 3. Similarly, the 3 units of the second job complete within the cutoff time of 3 and the third job is completed ahead of the cutoff time on time step 2. Summing the columns of matrix  $\mathbf{C}$ , we see that the maximal capacity of  $V = 2$  is fully utilized on each time step.

In order to evaluate the effectiveness of a given schedule  $\mathbf{C}$ , we define tardiness of a job as follows:

$$T_i = \max(0, F_i - K_i), \quad (2)$$

where  $F_i$  is the actual finish time of job  $i$  under schedule  $\mathbf{C}$ :  $F_i = \arg \max_j \{C_{i,j} = 1\}$  (The position of the last 1 in the  $i$ th row in scheduling matrix  $\mathbf{C}$ .)

The problem can now be stated formally as follows:

$$\mathbf{C}_{\text{opt}} := \arg \min_{\mathbf{C}} \sum_{i=1}^N w_i T_i. \quad (3)$$

Under the following constraints:

- The sizes of the scheduled jobs in the scheduling matrix are equal to the predefined amounts:

$$\sum_{j=1}^L C_{i,j} = x_i, \forall i = 1, \dots, N. \quad (4)$$

- The number of scheduled jobs at any given time instant does not exceed the capacity of the system:

$$\sum_{i=1}^N C_{i,j} \leq V, \forall j = 1, \dots, L. \quad (5)$$

We revisit our previous example with a minor change:  $V = 2$ ,  $N = 3$ ,  $\mathbf{x} = \{2, 3, 2\}$ ,  $\mathbf{K} = \{3, 3, 3\}$  and a weight vector  $\mathbf{w} = \{3, 2, 1\}$ . It can be observed that there is no solution, in which all jobs are completed before their cutoff times. A minimal weighted tardiness solution is the following:

$$\mathbf{C} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}.$$

### 3 Existing heuristic methods

Given the NP-hardness of the scheduling task as shown by Du et al. [7], and therefore the amount of time it would take to find the exact, optimal solution, in most real-world settings the pragmatic approach of finding a fast, sub-optimal, but good solution is followed. As outlined in the introduction, this has been a very active field of research, and most studies use the Earliest Due Date (EDD) and Weighted Shortest Process Time (WSPT) heuristics as benchmarks for evaluating the proposed algorithms. In addition to these, we also outline the recently developed Load Balancing Scheduling (LBS) heuristic and a newly proposed Largest Weighted Process First (LWPF) heuristic. Furthermore, we also outline a random processing method which is used as a low benchmark for our testing results.

#### 3.1 Earliest due date (EDD) heuristic

The EDD heuristic orders the sequence of jobs to be executed from the job with the earliest due date to the job with the latest due date. Using the notation of Section 2, we relabel the job indices so that the following inequality holds:

$$K_1 \leq K_2 \leq \dots \leq K_N.$$

Once this ordering is determined, the jobs are allocated to the machines in this order, always utilizing the maximum available capacity. Once a job finishes and capacity is freed up, the next job using the above ordering is scheduled on the freed up machine. It has been shown in [22] that EDD finds the optimal schedule when one wants to minimize the maximum tardiness on a single machine. However, we note that when the objective function includes the relative priorities of jobs, EDD is at a severe disadvantage, as it does not include consideration of the weights.

#### 3.2 Weighted shortest processing time (WSPT) heuristic

The WSPT method is analogous to the EDD method in that it orders the jobs according to well-defined criteria and then schedules the jobs according to this ordering, utilizing the maximum available capacity available. Once a job finishes and capacity is freed up, the next job using the ordering is scheduled. Using the notation of Section 2, the jobs are ordered such that the below holds:

$$\frac{x_1}{w_1} \leq \frac{x_2}{w_2} \leq \dots \leq \frac{x_N}{w_N}.$$

### 3.3 Largest weighted process first (LWPF) heuristic

This heuristic is analogous to the EDD and the WSPT heuristics with the jobs ordered according to the following inequality:

$$w_1 \geq w_2 \geq \dots \geq w_N.$$

This is a simple heuristic which allows us get a sense of the importance of considering the weights versus the cutoff times. We have not seen this simple heuristic mentioned anywhere in the literature, but it turns out to have quite good empirical characteristics which is one of the many contributions of this paper.

### 3.4 Load Balancing Scheduling (LBS) algorithm

Laszlo et al [12] suggest a novel deterministic, polynomial time algorithm for scheduling jobs with cutoff times, in the context of load balancing for wireless sensor networks. The basic idea of the algorithm is to start scheduling the jobs backwards from the maximal cutoff time, in order of decreasing cutoff times, always ensuring full capacity is utilized. For a more formal definition of the algorithm, please see [12]. We note that whilst the algorithm has proven to be extremely successful for load balancing, it does not take into account the weights of the jobs and therefore will suffer from the same shortcomings in our setting as EDD.

### 3.5 Random method

In this method, the jobs are ordered randomly and are scheduled in such a way as to always fully utilize the maximum available capacity. This unsophisticated heuristic is useful as a low benchmark by repeating it a number of times and taking the best solution over the multiple repeats.

## 4 Novel Hopfield neural network (HNN) based approach

Since the number of possible binary matrices grows exponentially with the number of nodes and the length of the schedule, exhaustive search with complexity  $O(2^{N \cdot L})$  is generally computationally infeasible to solve the optimization problem at hand. Thus, our goal is to develop a polynomial time approximate solution by mapping the problem to an analogous quadratic optimization

problem, similar to the approach of Levendovszky et al [15] and Treplan et al [27].

We first review the methods available for optimizing equations in quadratic form. Let us assume that matrix  $\mathbf{W}$  is a symmetric matrix of size  $n \times n$  and vector  $\mathbf{b}$  is of length  $n$ . We seek the optimal  $n$  dimensional vector  $\mathbf{y}$  which minimizes the following quadratic function [21]:

$$f(\mathbf{y}) = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} + \mathbf{b}^T\mathbf{y}, \quad (6)$$

subject to one or more constraints of the form of

$$\begin{aligned} \mathbf{A}\mathbf{y} &\leq \mathbf{v}, \\ \mathbf{B}\mathbf{y} &= \mathbf{u}. \end{aligned}$$

If the problem at hand contains only linear constraints then it can be solved as presented by Murty et al [20]. In other cases, if the matrix  $\mathbf{W}$  is positive definite, then the function  $f(\mathbf{y})$  is convex and the problem can be solved with the ellipsoid method presented by Zhi-Quan et al [28]. When  $\mathbf{W}$  is indefinite, the problem is NP-hard (for details see [24]).

A frequently used powerful, heuristic algorithm to solve quadratic optimization problems is the Hopfield Neural Network (HNN). This neural network is described by the following state transition rule:

$$\mathbf{y}_i(k+1) = \text{sgn} \left( \sum_{j=1}^N \hat{\mathbf{W}}_{ij} \mathbf{y}_j(k) - \hat{\mathbf{b}}_i \right), i = \text{mod}_N k,$$

where

$$\begin{aligned} \mathbf{d} &= -\text{diag}(\mathbf{W}), \\ \hat{\mathbf{W}} &= -\mathbf{W} - \text{diag}(\mathbf{d}), \\ \hat{\mathbf{b}} &= \mathbf{b} - \frac{1}{2}\mathbf{d}. \end{aligned}$$

Using the Lyapunov method, Hopfield [10] proved that the HNN converges to its fixed-point, as a consequence the HNN minimizes a quadratic Lyapunov function:

$$\mathcal{L}(\mathbf{y}) := -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \hat{\mathbf{W}}_{ij} \mathbf{y}_i \mathbf{y}_j + \sum_{i=1}^N \mathbf{y}_i \hat{\mathbf{b}}_i = -\frac{1}{2}\mathbf{y}^T\hat{\mathbf{W}}\mathbf{y} + \hat{\mathbf{b}}^T\mathbf{y}.$$

Thus, the HNN is able to solve combinatorial optimization problems in polynomial time under special conditions as it has been demonstrated by Mandziuk



[18, 17]. Using this method in practice we have to handle the following problem: in some cases the HNN method may get stuck in local minimal point of its Lyapunov function.

These methods motivate us to map the existing optimization problem into quadratic form. First the binary scheduling matrix  $\mathbf{C}$  is mapped into a binary column vector  $\mathbf{y}$  as follows:

$$\mathbf{C} = \begin{pmatrix} C_{1,1} & C_{1,2} & \cdots & C_{1,L} \\ C_{2,1} & C_{2,2} & \cdots & C_{2,L} \\ \vdots & \vdots & \ddots & \vdots \\ C_{N,1} & C_{N,2} & \cdots & C_{N,L} \end{pmatrix} \rightarrow$$

$$\mathbf{y} = (C_{1,1}, C_{1,2}, \dots, C_{1,L}, C_{2,1}, \dots, C_{2,L}, C_{N,1}, \dots, C_{N,L})^T.$$

The original objective function (3) is elaborated as follows:

$$\min_{\mathbf{C}} \sum_{i=1}^N w_i \left( \max \left( 0, \arg \max_j \{C_{i,j} = 1\} - K_i \right) \right).$$

This objective is equivalent to:

$$\min_{\mathbf{C}} \sum_{i=1}^N w_i \left( \sum_{j=K_i+1}^L C_{i,j} \right).$$

The minimization problem is thus equivalent to:

$$\mathbf{C}_{\text{opt}} := \arg \min_{\mathbf{C}} \sum_{i=1}^N \sum_{j=K_i+1}^L w_i C_{i,j}^2.$$

Therefore the mapping required is:

$$-\frac{1}{2} \mathbf{y}^T \mathbf{W}_A \mathbf{y} + \mathbf{b}_A^T \mathbf{y} = \sum_{i=1}^N \sum_{j=K_i+1}^L w_i C_{i,j}^2.$$

The solution is the following:

$$\mathbf{b}_A = \mathbf{0}_{NL \times 1},$$

$$\mathbf{W}_A = -2 \begin{pmatrix} \mathbf{D}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_2 & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{D}_N \end{pmatrix},$$

where

$$\mathbf{D}_j = \begin{pmatrix} \mathbf{0}_{K_j \times K_j} & \mathbf{0}_{K_j \times (L-K_j)} \\ \mathbf{0}_{(L-K_j) \times K_j} & w_j \mathbf{I}_{(L-K_j) \times (L-K_j)} \end{pmatrix} \in \mathbb{R}^{L \times L}.$$

Having transformed the objective function to a quadratic form, we now turn our attention to doing the same with the two constraints outlined in (4) and (5).

The constraints in (4) can be rewritten as follows:

$$\forall i : \sum_{j=1}^L C_{i,j} = x_i \rightarrow \min_{\mathbf{C}} \sum_{i=1}^N \left( \left( \sum_{j=1}^L C_{i,j} \right) - x_i \right)^2.$$

This will lead to the following equation:

$$-\frac{1}{2} \mathbf{y}^T \mathbf{W}_B \mathbf{y} + \mathbf{b}_B^T \mathbf{y} = \sum_{i=1}^N \left( \left( \sum_{j=1}^L C_{i,j} \right) - x_i \right)^2. \quad (7)$$

The solution of equation (7) is:

$$\mathbf{b}_B = 2 \begin{pmatrix} x_{1 \times L} & x_{2 \times L} & \cdots & x_{N \times L} \end{pmatrix},$$

$$\mathbf{W}_B = -2 \begin{pmatrix} \mathbf{1}_{L \times L} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{1}_{L \times L} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{1}_{L \times L} \end{pmatrix}.$$

Another constraint ensures that the scheduling does not overload the processing units, described in (5). The required transformation is as follows:

$$\forall j : \sum_{i=1}^N C_{i,j} = V \rightarrow \min_{\mathbf{C}} \sum_{j=1}^L \left( \left( \sum_{i=1}^N C_{i,j} \right) - V \right)^2. \quad (8)$$

Please note the technicality that this constraint may not be relevant for the last few columns where only the remaining jobs have to be scheduled and schedule matrices not exhausting the full capacity should not be penalized. The total length of scheduling can be written as follows:

$$\tilde{L} = \left\lceil \frac{\sum_{i=1}^N x_i}{V} \right\rceil,$$

$$L = \max \left( \tilde{L}, \max_i x_i, \max_i K_i \right).$$

The number of columns where capacity  $V$  needs to be fully utilized is the following:

$$M = \max \left( 1, \max_i x_i - \hat{L} + 1 \right). \quad (9)$$

Taking into consideration (9) the mapping of (8) can be described by the following equation:

$$-\frac{1}{2} \mathbf{y}^T \mathbf{W}_B \mathbf{y} + \mathbf{b}_B^T \mathbf{y} = \sum_{i=1}^M \left( \left( \sum_{j=1}^N C_{i,j} \right) - V \right)^2.$$

The solution of this equation is the following:

$$\mathbf{b}_C = [\mathbf{V}_{M \times 1}, \mathbf{0}_{(L-M) \times 1}, \mathbf{V}_{M \times 1}, \mathbf{0}_{(L-M) \times 1}, \dots, \mathbf{V}_{M \times 1}, \mathbf{0}_{(L-M) \times 1}],$$

$$\mathbf{W}_C = -2 \begin{pmatrix} \mathbf{D} & \mathbf{D} & \cdots & \mathbf{D} \\ \mathbf{D} & \mathbf{D} & \cdots & \mathbf{D} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{D} & \mathbf{D} & \cdots & \mathbf{D} \end{pmatrix},$$

where

$$\mathbf{D} = \begin{pmatrix} \mathbf{I}_{M \times M} & \mathbf{0}_{M \times (L-M)} \\ \mathbf{0}_{(L-M) \times M} & \mathbf{0}_{(L-M) \times (L-M)} \end{pmatrix}.$$

We can combine these mappings into the form of equation (6) as follows:

$$\mathbf{W} = \alpha \mathbf{W}_A + \beta \mathbf{W}_B + \gamma \mathbf{W}_C \in \mathbb{R}^{NL \times NL},$$

and

$$\mathbf{b} = \alpha \mathbf{b}_A + \beta \mathbf{b}_B + \gamma \mathbf{b}_C \in \mathbb{R}^{NL \times 1}.$$

Note that the objectives can be controlled with heuristic constants  $\alpha, \beta$  and  $\gamma$  in order to strike an appropriate balance between the weights of different requirements. Having this quadratic form at hand, we can apply the HNN to provide an approximate solution to the optimization problem in polynomial time.

## 5 Numerical results

In this section, the performance of the HNN approach is investigated and is compared to the performance of other heuristics.

### 5.1 Simulation method

Each method outlined in Section 3 and the HNN method outlined in Section 4 has been tested by simulation on a large and diverse set of input parameters with the aim to characterize the algorithms empirically on scheduling problems of different size of jobs. The algorithms were implemented in Matlab and tests were run in this simulation environment with randomly generated parameters such as size of jobs, cutoff times, and weights. The size of each job and its cutoff time and corresponding weight are generated as follows:

$$\mathbf{x}_i = \text{random}([1, c_1]), \quad (10)$$

$$\mathbf{K}_i = \mathbf{x}_i + \text{random}([c_1, 1.5 \cdot c_1]), \quad (11)$$

$$\mathbf{w}_i = \text{random}([1, c_2]), \quad (12)$$

where  $\text{random}(\Theta)$  produces a uniformly distributed random integer value in range  $\Theta$ . In our simulations the constant  $c_1$  equals 10 and  $c_2$  equals 5. The number of processors ( $V$ ) is determined as follows:

$$V = 0.25 \cdot J. \quad (13)$$

The problem is expected to be solved without tardiness when the capacity is  $V = J$ . Therefore (13) ensures that there is a high likelihood of tardiness associated with the generated problem.

For each problem size, 100 different problems were generated randomly, using (10)-(13) and the results the methods were compared in each case.

The random method was repeated 1000 times for each problem and the best solution was used. Furthermore, the HNN method was repeated 1000 times for each problem with different random starting points and the best solution was used. In addition, the heuristical parameters  $\alpha, \beta$  and  $\gamma$  were adjusted between the simulations in order to provide a good balance between optimizing the objective function, but also meeting the required constraints to produce a valid scheduling matrix. To adjust the heuristical parameters we used Algorithm 1.

Even so, in some cases the HNN is unable to provide a valid solution, because the prescribed constraints on job sizes and capacity are violated. Therefore, we introduced an error correction function (see Algorithm 2) in order to cut and replace the unnecessary 1-s in the scheduling matrix. In our simulations parameter  $e$  equals 5. All of the results presented in the following sections concern valid schedules meeting the required constraints.

---

**Algorithm 1** Algorithm for adjusting the heuristical parameters

---

**Require:**  $\mathbf{x}, \mathbf{K}, V, e$   
 $\alpha \leftarrow 0.1, \beta \leftarrow 5, \gamma \leftarrow 5$   
 $i \leftarrow 0$   
**repeat**  
   $i \leftarrow i + 1$   
   $\mathbf{C}_i \leftarrow \text{HNN}(\mathbf{x}, \mathbf{K}, V, \alpha, \beta, \gamma)$   
   $\alpha \leftarrow \alpha + 0.01$   
**until**  $\text{errors}(\mathbf{C}_i) \leq e$   
**for**  $k = 1 \rightarrow i$  **do**  
   $\mathbf{C}_k \leftarrow \text{correct}(\mathbf{C}_k)$   
   $\mathbf{T}_k \leftarrow \text{calculateTWT}(\mathbf{C}_k)$   
**end for**  
**return**  $\min(\mathbf{T})$

---



---

**Algorithm 2** Correction algorithm for the scheduling matrix produced by the HNN

---

**Require:**  $\mathbf{x}, \mathbf{w}, \mathbf{K}, V, \mathbf{C}$   
**for**  $k = 1 \rightarrow L$  **do**  
  **while**  $\sum_{i=1}^N \mathbf{C}_{i,k} > V$  **do**  
    Remove 1 from row  $j$ , where  $j$  is the row in column  $k$  with minimal weight that has  $\mathbf{C}_{j,k} = 1$   
  **end while**  
**end for**  
**for**  $k = 1 \rightarrow N$  **do**  
  **while**  $\sum_{i=1}^L \mathbf{C}_{k,i} > \mathbf{x}_k$  **do**  
    Remove 1 from row  $k$  from the column  $l$ , where  $l$  is the righternmost column in row  $k$  such that  $\mathbf{C}_{k,l} = 1$   
  **end while**  
  **while**  $\sum_{i=1}^L \mathbf{C}_{k,i} < \mathbf{x}_k$  **do**  
    Add 1 to row  $k$  in column  $l$  where  $l$  is the lefternmost column where 1 can be added without violating the capacity constraint  $V$ .  
  **end while**  
**end for**  
**return**  $\mathbf{C}$

---

## 5.2 Average total weighted tardiness of the different methods

The first simulation compares the average total weighted tardiness provided by the algorithms for different problem sizes.

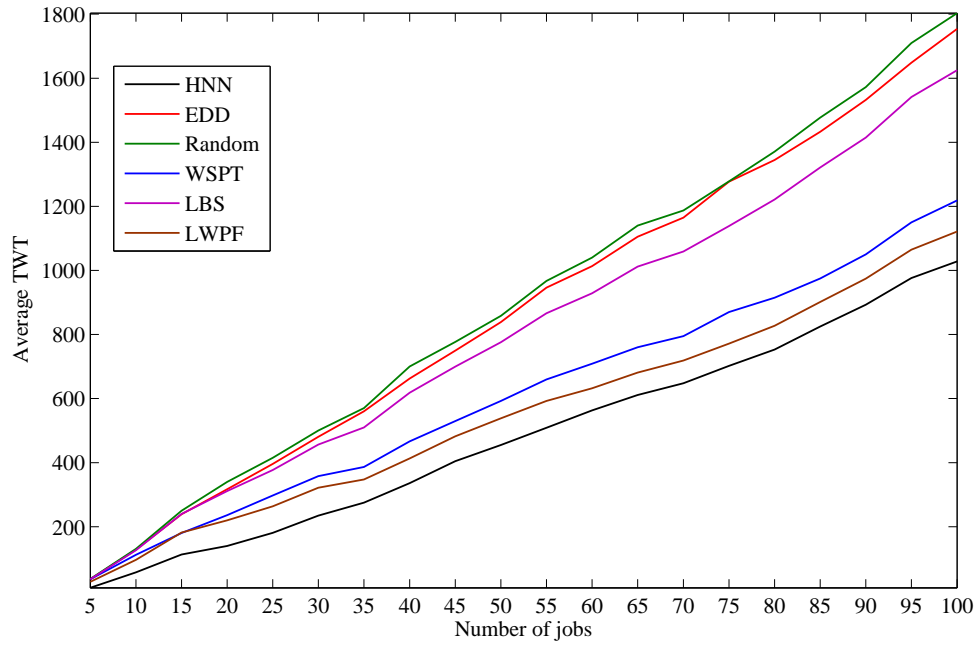


Figure 1: Average TWT produced by each heuristic over randomly generated problems depicted as a function of the number of jobs in the problem.

Figure 1 demonstrates that the best solution achieved by the HNN produces better average TWT for all problem sizes than any of the other heuristics. The performances of the EDD, LBS and random methods are, on average, worse than the WSPT and LWPF heuristics and the HNN for all problem sizes. This shows that consideration of the weights in the optimization problem is critical to reach near-optimal TWT schedules. It is also clearly visible that the HNN consistently outperforms all other methods for all problem sizes. Note also that the simple LWPF heuristic, proposed by us, outperforms the WSPT heuristic widely used as a benchmark in the literature.

### 5.3 Detailed comparison of the HNN performance versus other heuristics

In this section, we quantify how much better the solution provided by the HNN is, in comparison to the EDD, WSPT and LWPF algorithms. Figure 2 depicts the ratio of average TWT produced by the HNN method versus the other heuristics. We conclude that, on average, the best solution of the HNN heavily outperforms the traditional solutions: the total weighted tardiness of the best HNN solution is 25% – 56% of the EDD, 25% – 84% of the WSPT, and 34% – 91% of the LWPF. As the problem size increases, the WSPT and LWPF heuristics produce solutions which are closer to the HNN.

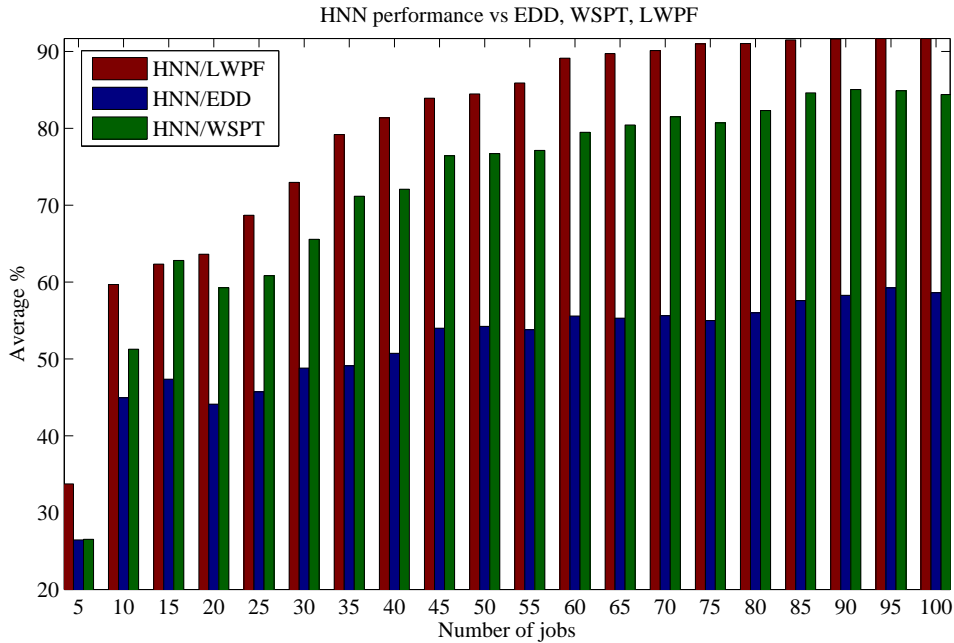


Figure 2: Ratio of average TWT produced by the HNN versus EDD, WSPT and LWPF as a function of the problem size

In order to verify that the HNN consistently outperforms the other heuristics on a wide spectrum of problems, not just on a few selected problems for each given problem size, we ran a more detailed experiment. In this case, we investigated 500 randomly generated problems for each problem size and computed the percentage of times the HNN produced a better solution than LWPF, the next best method. Table 1 shows the result which quite convinc-

ingly proves the general applicability of the HNN to different problem types. Across the spectrum of problem domain, the ratio is higher than 98.5% for all investigated job sizes.

Table 1: Percentage of problems in which the HNN provides an improved solution over the next best heuristic, LWPF

Job size	5	10	20	25	50	75	100
	99.9%	100%	99.5%	99.2%	99.3%	98.6%	98.8%

#### 5.4 Runtime characteristics of the investigated algorithms

In this section, we summarize the runtime characteristics of the introduced algorithms. Table 2 contains the theoretical order of convergence of the algorithms as a function of the length of the input parameters.

Table 2: Theoretical order of convergence of the investigated algorithms

Algorithm	Order of convergence	Reference
Random strategy	$O(L \cdot N)$	[12]
EDD, WSPT, LWPF	$O(L \cdot N)$	[16]
LBS	$O(L \cdot N^2)$	[12]
HNN	$O(L^2 \cdot N^2)$	[10], [9]
Exhaustive search	$O(2^{L \cdot N})$	

Note that this table shows the theoretical runtime for a single run of the specified algorithm, whilst we need to run a constant number of the HNN and random iterations. On the other hand, these independent runs can be executed in parallel on several computers, multicore machines or even GPU-based architectures, so the overall runtime should not be significantly different due to the need to run multiple iterations.

Figure 3 depicts the empirical runtime comparison of the different heuristics which confirm the theoretical limits. Although the HNN is the slowest method of the investigated heuristics, in practice we see that near optimal solution of a 100 job problem takes only around 6 seconds / iteration (including the iterative adjustment of the heuristic parameters as per Algorithm 1) on Core i7@3Ghz processor, running non-optimized Matlab code, which is very promising as far as its practical applicability in real life applications is concerned. With further optimization and parallelization on multicore machines or GPGPU technology,



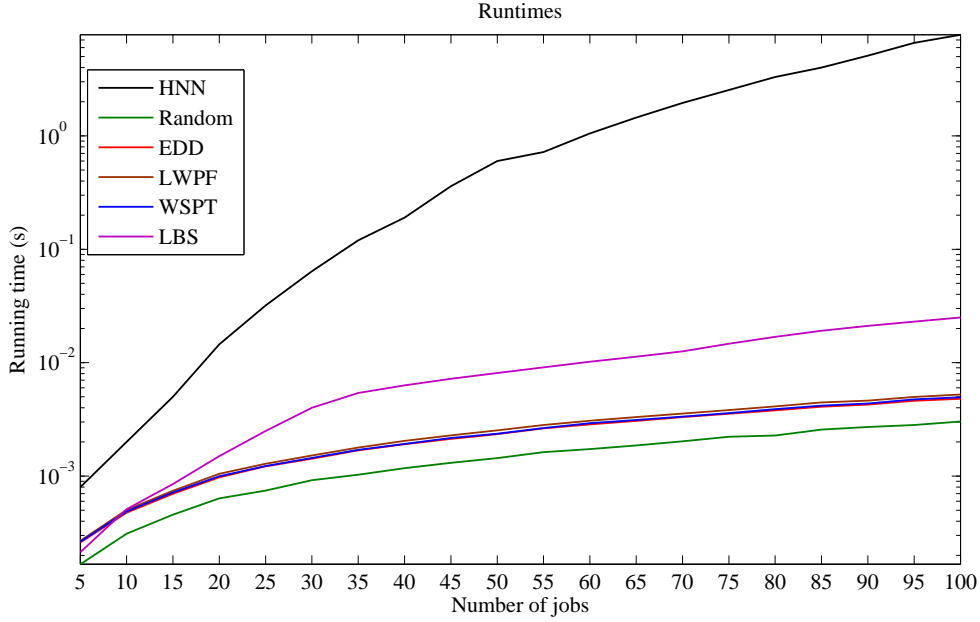


Figure 3: The average runtime of each algorithm per iteration (including the iterative adjustment of the heuristic parameters as per Algorithm 1 for HNN), depicted as a function of the number of jobs. Note that the vertical axis is using a logarithmic scale

the HNN method provides better quality solutions than the other heuristics within acceptable runtimes.

## 6 Conclusions and directions of future research

In this paper, we studied the NP-hard problem of scheduling jobs with given relative priorities (weights) and deadlines on identical machines, minimizing the TWT measure. We developed a novel heuristic approach, utilizing quadratic programming and the Hopfield neural network and we showed that, in general, it outperforms the EDD and WSPT methods. Furthermore, we have shown that our approach can be applied in real-time settings for small problem sizes and possesses scalability features which are acceptable for real world applications.

More formal methods for the selection of alpha, beta and gamma parameters

could be investigated in the future to further improve performance. Another idea to explore is whether a more intelligent selection of the initial point for the HNN algorithm (eg. the result of the LWPF or WSPT heuristics) would improve the algorithms performance over the random starting point which was used in our tests. Finally, the performance of the HNN algorithm needs to be compared to more complex heuristic methods.

## References

- [1] D. E. Akyol, G. M. Bayhan, Multi-machine earliness and tardiness scheduling problem: an interconnected neural network approach. *Int. J. of Adv. Manuf. Technol.*, **37** (2008) 576–588. [⇒ 50](#)
- [2] V. A. Armentano, D. S. Yamashita, Taboo search for scheduling on identical parallel machines to minimize mean tardiness. *J. of Intell. Manuf.*, **11** (2000) 453–460. [⇒ 50](#)
- [3] M. Azizoglu, O. Kirca, Tardiness minimization on parallel machines. *Int. J. of Production Econ.*, **55** (1998) 163–168. [⇒ 50](#)
- [4] D. Biskup, J. Herrman, J. N. D. Gupta, Scheduling identical parallel machines to minimize total tardiness. *Int. J. of Production Econ.*, **115** (2008) 134–142. [⇒ 50](#)
- [5] P. Brucker, *Scheduling Algorithms*, 5th edn. New York: Springer, 2007. [⇒ 49, 51](#)
- [6] A. Dogramaci, J. Surkis, Evaluation of a heuristic for scheduling independent jobs on parallel identical processors. *Manag. Sci.*, **25**, 12 (1979) 1208–1216. [⇒ 50](#)
- [7] J. Du, J. Y. T. Leung, Minimizing total tardiness on one machine is np-hard. *Math. Oper. Res.*, **15**, 3 (1990) 483–495. [⇒ 50, 53](#)
- [8] A. Guinet, Scheduling independent jobs on uniform parallel machines to minimize tardiness criteria. *J. of Intell. Manuf.*, **6** (1995) 95–103. [⇒ 50](#)
- [9] S. Haykin, *Neural Networks and Learning Machines*, 3rd edn. Prentice Hall, 2008. [⇒ 63](#)
- [10] J. J. Hopfield, Neural networks and physical systems with emergent collective computational abilities. In: *Proc. Natl. Acad. Sci. USA*, **79** (1982) 2554–2558. [⇒ 55, 63](#)
- [11] C. P. Koulamas, The total tardiness problem: review and extensions. *Oper. Res.*, **42** (1994) 1025–1041. [⇒ 50](#)
- [12] E. Laszlo, K. Tornai, G. Treplan, J. Levendovszky, Novel load balancing scheduling algorithms for wireless sensor networks. In: *The Fourth Int. Conf. on Communication Theory, Reliability, and Quality of Service*, Budapest, 2011, pp. 54–49. [⇒ 54, 63](#)
- [13] E. L. Lawler, A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. *Ann. of Discrete Math.*, **1** (1977) 331–342. [⇒ 50](#)

- [14] E. L. Lawler, Preemptive scheduling of precedence-constrained jobs on parallel machines, deterministic and stochastic scheduling. *Proc. NATO Advanced Study and Research Institute on Theoretical Approaches to Scheduling Problems*, 1981, pp. 101–123. [⇒ 49](#)
- [15] J. [Levendovszky](#), A. Olah, K. [Tornai](#), G. Treplan, Novel load balancing algorithms ensuring uniform packet loss probabilities for WSN, *IEEE 73rd Vehicular Technology Conf., Budapest*, 2011. [⇒ 55](#)
- [16] R. Maheswaran, S. G. Ponnambalam, S. D. Nithin, A. S. Ramkumar, Hopfield neural network approach for [single machine scheduling problem](#). In: *Proc. 2004 IEEE Conf. on Cybernetics and Intelligent Systems*, 2004, pp. 850–854. [⇒ 50, 63](#)
- [17] J. [Mandziuk](#), Solving the [travelling salesman problem](#) with a Hopfield – type neural network. *Demonstratio Math.*, **29**, 1 (1996) 219–231. [⇒ 56](#)
- [18] J. [Mandziuk](#), B. [Macukow](#), A neural network designed to solve the [n-queens problem](#). *Biol. Cybernet.*, **66**, 4 (1992) 375–379. [⇒ 56](#)
- [19] C. [Martel](#), [Preemptive Scheduling](#) with Release Times, Deadlines, and Due Times *J. of the ACM*, **29**, 3 (1982) 812–829. [⇒ 49](#)
- [20] K. G. [Murty](#), Linear complementarity, [linear and nonlinear programming](#). Heldermann Springer-Verlag, 1988. [⇒ 55](#)
- [21] J. Nocedal, S. J. Wright, *Numerical Optimization*. Springer-Verlag, 1988. [⇒ 55](#)
- [22] M. [Pinedo](#), *Scheduling – Theory, Algorithms and Systems*, 3rd edn. New York: Springer, 2008. [⇒ 49, 53](#)
- [23] R. M. V. Rachamadugu, T. E. Morton, *Myopic heuristics for the weighted tardiness problem on identical parallel machines*. Tech. rep., (CMU-RI-TR-83-17) Carnegie-Melon University, the Robotics Institute, 1983. [⇒ 50](#)
- [24] S. [Sahni](#), [Computationally related problems](#). *SIAM J. on Comp.* **3** (1974) 262–279. [⇒ 55](#)
- [25] S. [Sahni](#), [Preemptive scheduling](#) with due dates. *Oper. Res.*, **27**, 5 (1979) 925–934. [⇒ 49](#)
- [26] T. Sen, J. M. [Sulek](#), P. [Dileepan](#), [Static scheduling research](#) to minimize weighted and unweighted tardiness: A state-of-the-art survey. *Int. J. of Production Econ.*, **83** (2003) 1–12. [⇒ 50](#)
- [27] G. Treplan, K. [Tornai](#), J. [Levendovszky](#), [Quadratic programming](#) for TDMA scheduling in wireless sensor networks, *Int. J. of Distrib. Sen. Netws.*, 2011. 17 p. [⇒ 55](#)
- [28] L. [Zhi-Quan](#), M. Wing-Kin, M. C. S. Anthony, Y. Yinyu, Z. Shuzhong, Semidefinite relaxation of [quadratic optimization](#) problems. *Signal Process. Mag.*, IEEE **27**, 3 (2010) 20–34. [⇒ 55](#)

*Received: March 5, 2012 • Revised: April 16, 2012*